DOCUMENT RESUME

ED 304 134 IR 013 707

AUTHOR Adamson, Eadie
TITLE Exploring Motion through Game MicroWorlds with LOGO: How Theory Transferred to the Classroom.
PUB DATE 88
NOTE 15p.; Paper presented at the Meeting of the New York State Association for Computers and Technologies in Education (Ryebrook, NY, November 20-22, 1988).
PUB TYPE Computer Programs (101) -- Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Games; Computer Software; *Discovery Learning; *Discovery Processes; *Grade 5; Intermediate Grades; Keyboarding (Data Entry); *Programing; *Teacher Student Relationship
IDENTIFIERS *LOGO Programing Language

ABSTRACT

This paper describes the use of LOGOWriter with fifth grade boys as a means of developing programming procedures for computer games that they could use to simulate motion with acceleration, as well as techniques for controlling the direction of motion. The kinds of programs developed, student responses to this work, observations about teaching issues which arose, and details about several different ways to approach the topic with a group of students are all considered. It was found that the agenda of developing reasonably accurate simulation of motion fitted neatly into projects that the students--who had several years of previous Logo experience--were already highly motivated to develop, and new ideas generated by the students permitted opportunities for student-teacher interaction. The extent to which students were, and could be, permitted to discover on their own and the extent to which teacher intervention was required is also discussed, and it is noted that two different approaches were used with different groups of students: a discovery method with students working from a target game, and the development of an individual game project by each student. The particular experiences of several individual students are also described. (EW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

Exploring Motion Through Game MicroWorlds With Logo: How Theory Transferred to the Classroom

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Eadie Adamson

Copyright 1988 Eadie Adamson

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Eadie Adamson
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

ED304134

Introduction

Two years ago, in the winter of 1986, Michael Tempel, Director of Educational Services for Logo Computer Systems, and I began observing and working with a group of fourth grade students at The Allen-Stevenson School, an independent school for boys in New York City. They had just begun using LogoWriter. The boys were fascinated with the possibilities of controlling speed and direction of objects moving across the screen. They had programmed scenery and were busy developing programs to "drive" a car or "fly" a helicopter or airplane. We saw the domain of motion as one ripe for development in a Logo environment. The student agenda involved moving objects around in some fairly complex ways, yet their ability to express this in Logo was limited. Accuracy of the motion simulation became our adult goal which had possibilities for connecting with the students' more flexible agenda of simply getting things to move.

We decided that during the next year we would provide programming pieces to extend the possibilities for what the students might plan to create in a motion and animation context. Our planning included developing procedures that simulate motion with acceleration and friction as well as techniques for controlling the direction of motion. We were curious about the students' response: whether they would use these procedures, whether through this experience they would gain any new understanding of the way real things move or the way in which such movement could be represented in a Logo program, and finally, whether they would appropriate these procedures in games they created themselves.

This paper is a brief report on what happened: the kinds of programs developed, student response to this work, observations about teaching issues which arose, and details about several different ways to approach such a topic with a group of students. The focus is on fifth grade students, ages ten and eleven, with several years of previous Logo experience as background.

A LogoWriter Game Microworld

During the 1987-1988 school year we had the opportunity to work with groups of students in half-year intervals during which classes met twice per week. All of

IR013707

these students had worked in the previous year on "driving" programs and already knew something about interactive control of motion. They had learned a little about program structure and knew how to place player cues in the command center, a technique they had used when building their earlier programs. Most of the boys also had programmed control keys to operate on speed of an object.

In the first term we began by working with the boys on writing a simple target game using a single turtle. Once this very simple program was in place, the students were encouraged to propose and work out, with assistance, variations on the "theme". The programming techniques introduced in the initial game included player input to turn the turtle and request a shot at the target, getting a distance estimate from the player for an input to a **forward** command expressed by **fd first readlistcc** , and, finally, introduction of the **colorunder** primitive to detect a hit. The simple game shell was open to adaptation with very little additional programming: random placement of a target and the turtle as well as changing from a static to a moving turtle, for example, was easily accomplished. The game could easily be changed to a dynaturtle game, picking up on the interest of students in space travel. Following the extensions of the game, many boys appropriated what they had learned and very independently incorporated the ideas in other games. Others focused more on embellishing the simple shell with touches of their own, such as imaginative introductions which set a context for the game.

Here is the basic game from which we began:

First, draw a target on the screen and send the turtle **home**:

```
TO TARGET
PU SETPOS [100 70] PD
REPEAT 4
      [FD 20 RT 90]
PU
RT 45
FD 15
PD
FILL
PU
HOME
END
```

Write a procedure which checks recursively for a key to be pressed and also checks to see if the turtle hits the target.

```
TO GO
IF KEY? [AIM READCHAR]
IF COLORUNDER = 1 [STOP]
GO
END
```

The line **IF KEY?** checks if it is **true** or **false** that a key was pressed. If a key was pressed, **GO** will call another procedure, using the key read from the Command Center (**readchar**) in the next procedure, **AIM**.

The color check, **IF COLORUNDER = 1 [STOP],** constantly checks if the turtle has hit the target. In this case, the target is white. If the turtle lands on white, the game ends. (If the color check is included in another procedure, it will be necessary to use **stopall** to halt the game. Using **stop** in a procedure called from a recursive procedure stops only that procedure and not the recursive procedure from which it is called.)

GO checks for the key and sends it off to another procedure, **AIM**:

```
TO AIM :KEY
IF :KEY = "L [LT 15]
IF :KEY = "R [RT 15
IF :KEY = "S [SHOOT]
END
```

Pressing **L** turns the turtle left; pressing **R** turns the turtle right. When **S** is pressed, **SHOOT** collects the distance estimate from the player and uses it as an input to **forward**:

```
TO SHOOT
TYPE [HOW FAR?]
TYPE CHAR 32
FD FIRST READLISTCC
END
```

(**Type** places the text in the command center; **type char** 32 adds a space after the question.)

After **AIM** finishes its task, the control returns to **GO**. **GO** checks if the turtle is on the target. If so, it stops the game. If the turtle is not on the target, **GO** waits for another key to be pressed.

The static target game can easily be converted to a game with a moving turtle.

Here are the changes for adding a moving turtle:

Eliminate the **SHOOT** procedure and the **IF :KEY = "S ...** in **AIM**.

Change **GO** to read:

```
TO GO
FD 1
IF KEY? [AIM READCHAR]
IF COLORUNDER = 1 [STOP]
GO
END
```

GO can also be written as a procedure with a variable for the "speed":

```
TO GO :DIS
FD :DIS
IF KEY? [AIM READCHAR]
IF COLORUNDER = 1 [STOP]
GO :DIS
END
```

From this game it is a very short step to working with a dynaturtle. First, write a **SETUP** procedure which initializes some values for the X distance and the Y distance of the turtle:

```
TO SETUP
NAME 0 "XDIS
NAME 0 "YDIS
END
```

Change **GO** to read as follows:

```
TO GO
IF KEY? [AIM READCHAR]
SETX XCOR + :XDIS
SETY YCOR + :YDIS
IF COLORUNDER = 1 [STOP]
GO
END
```

Change **AIM**:

```
TO AIM :KEY
IF :KEY = "L [LT 30]
IF :KEY = "R [RT 30]
IF :KEY = "K [KICK]      <--This line is added
END
```

Define **KICK** as follows:

```
TO KICK
NAME (:XDIS + SIN HEADING) "XDIS
NAME (:YDIS + COS HEADING) "YDIS
END
```

Then try turning and kicking to get the turtle to the target.

Some Other Extensions

Many of the boys wanted to use the arrow keys to control the turtle. By typing **SHOW ASCII READCHAR** , pressing RETURN, and then pressing an arrow key, one can get the ASCII value of the key in the command center. In the **AIM** procedure one can then include arrow keys to turn the turtle. When an arrow key is pressed, the moving object's heading can be set in the corresponding direction (0, 90, 180 or 270). For example, the right arrow key can be programmed like this:

```
IF :KEY = CHAR 21 [SETH 90]
```

One caution here: use the ASCII values only for keys which cannot be accessed directly; do not use the ASCII values for letter or number or symbols.

Another method of controlling the motion does not require the kind of **IF KEY?** structure outlined above. The control keys, of which LogoWriter has ten, can be used to operate on direction and speed of the motion. These programmed event keys interrupt the process in a manner to similar to the interruption when one **ASK**s a turtle to do something. In some cases using control keys seems preferable to checking for keypresses, particularly if the program itself eventually has a large number of checks for position, color, score, time and so on. Not only is the program less cluttered when control keys are used to act on turtle movement, but it should run a little more smoothly.

The ten keys which can be programmed are:

N O P Q R V W X Y Z

When programmed, pressing CONTROL and the key will produce the desired result. In a game, these keys need to be activated at the start of the game. Such a procedure might look like this:

```
TO KEYS
WHEN "Z [NAME :SPEED + 1 "SPEED]      (This increases the speed.)
WHEN "Y [SETH 90]    (The turtle's heading turns directly right.)
END
```

When multiple turtles are used, the programmed keys should use **ASK** and the turtle number, followed by the command or commands enclosed in brackets, like this:

```
WHEN "Z  [ASK 0 [NAME :SPEED + 1 "SPEED] ]
```

Some Teaching Issues

Use of the discovery method is often interpreted as meaning never giving the students an answer but instead allowing (or requiring) them to find things out for themselves. There is much in Logo, however, that is not discoverable unless one has access to numerous manuals and instruction books. Giving students a model which makes central ideas clear and then encourages exploration is in step with the Logo philosophy of learning.

What we found, as we "invented" with each group, was that working through the process led to increased understanding. This was evidenced by later appropriation of the programming techniques in other games. A small exercise such as this afforded many good teaching opportunities. The sure knowledge that everyone in the group was grasping the basic concepts in part made up for the overly structured thrust and provided everyone with a good foundation on which to build in the following year. Perhaps the fact that we were open to taking advantage of a boy's idea for a change and were willing to jump off with him in another direction made the project successful. Eventually a wide number of variations were developed on this very simple beginning theme.

In fact, we were not exactly "inventing the wheel" since we were working out a structure which required introduction of Logo terms unknown to the students. We were giving the students a model from which they could learn. For example, while working together, each individual was exploring how such terms as **colorunder**

worked or did not work under the limited conditions of a microworld we were building ourselves. The boys began to see that a "fast" turtle, characterized by huge forward commands of 50 or 100, made **colorunder** ineffective. They began to choose the relatively slow "speeds" under 10 and began to regard with some disdain their fellow classmates who persisted in large numbers. As one boy said, "What's the point of using 100? It's just silly. Try five or ten. Ten is actually pretty fast!" The speed, he was saying, is relative to what you are trying to do. If you are detecting a target, even a turtle moving at forward 10 can present a challenge to a player. Setting a speed of 100 turns it into a game, not of skill, but of chance, since the turtle can only detect the color underneath when it is at rest. Testing for color in increments of 10 or less is clearly move effective than increments of 100.

I began to think of this issue in terms of my own learning. While I like to respond to challenges, I use all the resources available to me: my past experience, books on Logo, articles related to what problems I am working on, and, what is especially valuable, samples of other programs. I may have a particular problem in mind, but I rarely place myself in the position of solving it unaided or uninformed. There is no reason for students to be asked to invent it all alone either.

There is good reason, then, to provide students with a framework to explore. It should not be something so perfect that nothing else is left to do with it, but rather something so simple that it invites them to change it to suit their own ideas. There should be room for their imagination to go to work. This is the kind of activity the simple target game provided. The number of variations and extensions of the idea attested to the success of this approach. In addition, some students' appropriation of the ideas and application of this to other game structures showed us clearly that the concepts were understood and accepted as useful.

Second Term: A Different Way to Begin

During the second half of the year, we tried another approach with one particularly strong group of programmers. Instead of working from a target game, these boys proposed individual game projects, planning them out and narrowing down the beginning problems. Introduction of necessary techniques was handled slightly differently. Occasional lecture-demonstrations were given of the new programming terms and techniques which would feed into student projects. Sometimes these sessions involved only a small pullout group who were directly involved with working on a particular problem. This was backed up by a number of

"how-to" sheets which could be used independently as needed. The fact that the boys had an independent resource, other than a teacher, empowered this group still further. The complexity of their programming grew, yet at the same time appropriation of earlier tools or procedures in a new game was much in evidence.

Working with the second group, whose motion and programming problems were more complex, we found we were able to be more active in anticipating and providing solutions for those difficult spots beyond their programming ability. Rather than allowing the boys to be mired in difficulty with a peripheral problem, we gave them tool procedures which solved the problem and which could be manipulated easily. This enabled the boys to focus on the larger problems involved in developing their game and to pursue their projects with less frustration and with more appreciation for programming style and technique.

How much help to give became a teacher decision. In each instance the question was weighed in relation to the student's level of knowledge, whether the problem was peripheral or central to his goal, whether the problem was a result of an idiosyncrasy of the programming environment, and whether "discovery" of the solution was truly possible. Giving the information clearly does not preclude appropriation by the user. The process of making knowledge one's own is what is most important. Individual boys began to develop multi-screen games. As they moved from one level of game to another, these boys also moved those useful procedures from their earlier efforts into the new project. We saw this happen repeatedly.

One Student's Learning

James was working on a LogoWriter version of Afterburner, a very popular video game. James, although new to Logo, rapidly began to "speak" the language but often did not know the correct syntax for what he wanted to do. When working with James, we would talk the problem out in English, often typing this on the **flip** side, moving our proposed action closer and closer to Logo syntax. Then I would supply the answer in Logo.

James' first problem was this: A large sun nearly fills the screen. A planet is to rotate horizontally around the sun. Setting the planet in motion in a horizontal direction looks okay at first, but really is not giving the appearance of rotating. It is wrapping the screen. We must imagine it passing behind the sun.

James wanted his player to <u>see</u> the planet orbiting rather than supply the pass behind the planet in his imagination. I showed James how to find the edges of the

screen. Together we worked out the maximum **xcor** on each side. We wrote a procedure which checked the **xcor**. If it was greater than 120 the heading of the planet was reversed to 270; if it was less than -120 the heading was reset to 90. This worked better, but the planet still passed in front of the sun as it moved in either direction.

James knew what needed to happen: when the planet was moving in one direction, it needed to be hidden when moving over the sun and to reappear when moving over the background. The programming solution was beyond James' ability at this stage, but once I was sure that he could express the problem and potential solution clearly IN ENGLISH, I provided him with the tool procedure which would accomplish the task:

```
TO CKPOS0
IF XCOR > 130 [SETH 270 STOP]
IF XCOR < (-125) [SETH 90]
IF AND HEADING = 90 (OR XCOR < 63 XCOR > -67) [HT]
END
```

The first two lines in this procedure simply change the direction of the planet when it reaches the edge of the screen. The third line was the one which gave James trouble. It says just what he intended: if the turtle is moving directly to the right and it is on top of the large sun drawn in the center of the screen, the turtle should be hidden. James had to find the X-coordinates for the edges of the sun. He already knew how to do this: place the turtle at the edge of the shape and type **SHOW XCOR**. The X-coordinate would appear in the command center. James supplied the coordinate numbers for the procedure himself. I simply gave him the Logo translation of his description of what was to happen.

Tough problems which required new learning for James, such as constructing an oval-shaped landing pad, were begun with experimental procedures with inputs for him to play with. As we worked together, we would stop to analyze what was happening, what looked right and what didn't look right. I made it into a joint problem-solving session so that, even though James had never tried drawing ovals before, he was the chief participant and was fully involved in the decisions and the experimenting.

As James moved from problem to problem, I began to see that there was a pattern developing, a method of working with him which was not only very productive but which led him to make a number of attempts to solve problems on his own. Often his syntax was not quite correct, but he would present me with the problem,

expressing it in "near-Logo" language. For example, to check two sets of coordinates, James first wrote something like this:

```
IF POS = [120 0] OR POS = [100 0] [STOP]
```

Clearly, he understood that he could check both positions at the same time. He even used the word **or**, indicating his understanding of that term: that what was to happen as a result depended upon one or the other of the positions being true, but not both. We had not yet done any lessons, formal or informal, on the use of **and** or **or**. I showed James how it should be expressed, praising him for his attempt and pointing out how close he was to correct Logo usage:

```
IF OR (POS = [120 0]) (POS = [100 0]) [STOP]
```

James' game developed into a multilevel multi-page game with many complex tests for position or color and position. As he continued to expand the game, he worked more and more independently, adapting earlier procedures to fit the new levels of play. This is precisely what I had hoped would happen: as understanding grew, James was able to appropriate previous work and adapt it to his own purposes.

Another Year, Another Approach

This year (1988-1989) we began working with a group of fifth grade students who had had just a little exposure to working with a target game on a very simple level. Since the older boys working in the spring term last year had encountered a number of problems with understanding how to control multiple turtles, I decided to focus on developing some idea of how to control the turtles before trying to write games or animations with this new group.

We began by writing a "dance" for a single turtle. The turtle should, if possible, begin and end at **home**. Once this is created, it is easy to introduce the four turtles. There is already something interesting programmed for the turtles to do. Introduction of **tell all**, **tell** and a list of turtles, and **each** fit naturally into this environment. As we explored turtle motion under these conditions, leaving the shape as a turtle, some of the boys wanted the turtle to move sideways without turning. I showed them how to write these procedures, explaining a little about the use of **xcor** as we did it:

```
TO SLIDE.R
SETX XCOR + 5
END

TO SLIDE.L
SETX XCOR - 5
END
```

This procedure and others for moving vertically up and down (operating on the Y-coordinates) can also be written with a variable input so that the "speed" can be controlled or changed within the program.

These procedures, which we called "SLIDES", eventually gave rise to a number of wonderful inventions as well as some further exploration as a group. The problem came up: "I want the turtle to turn AND keep moving in one direction. How can I do that?" This is an extension of the sideways move:

```
SLIDE.R RT 10
```

will move a turtle sideways and, if repeated 36 times will also turn it completely around. Moving one turtle in this fashion while another moves in a single straight line involves direct confrontation with the issues of parallelism in the real world and the serial nature of the computer. It required us to think and talk about breaking movements down into their smallest elements to give a fairly successful simulation of parallelism. A procedure to move two turtles the same distance and direction, with only one turtle turning, broken into its smallest parts, would look like this:

```
TO SLIDE.TURN
TELL 0
SLIDE.R RT 10
TELL 1
SLIDE.R
END
```

Each turtle moves in the same increment; only turtle 0 turns. Adding a variable :TIMES to SLIDE.TURN makes the procedure more useful:

```
TO SLIDE.TURN :TIMES
REPEAT :TIMES
      [TELL 0 SLIDE.R RT 10
       TELL 1 SLIDE.R]
END
```

Thinking together about problems of this kind is excellent preparation for more complex programming and provides an interesting prelude to working with a dynaturtle, since the idea of changing movement by changing the coordinates is already in place.

One boy, Jim, has already begun work on "driving" a car. Although we showed him how to use **repeat 50 [fd 2]** to get a car moving, when he began programming on his own he took a totally different approach. Instead Jim copied the **SLIDE** procedures to his new page, at once demonstrating how well he comprehended what these procedures did, and began to program his driving program this way:

```
TO SETUP
SETSH 27
PU
SETH 90
END

TO DRIVE
SLIDE.R
DRIVE
END
```

The **seth 90** in the **SETUP** procedure is actually unnecessary in this context, since we are not turning the turtle at all. What is interesting, though, is the direct appropriation of the idea of the **SLIDE** as a state-change operator (although Jim would never describe it as such).

Jim, who has just begun this project, has decided to use a diagonal **SLIDE** to move his car from lane to lane on the highway he has drawn. He has turned his **SLIDE** procedures into procedures with variables in order to control "speed". The next step will be to incorporate a simulation of friction into this method of moving an object.

Jim's diagonal slide up to the right looks like this:

```
TO SLIDEUP.R :SPEED
SETX XCOR + :SPEED
SETY YCOR + :SPEED
END
```

His slide down right is like this:

```
TO SLIDEDOWN.R :SPEED
SETX XCOR + :SPEED
SETY YCOR - :SPEED
END
```

If Jim uses keys to control direction, he will change DRIVE:

```
TO DRIVE :SPEED
SLIDE.R :SPEED
IF KEY? [DO READCHAR]
DRIVE :SPEED
END
```

Jim's DO procedure will then read like this:

```
TO DO :KEY
IF :KEY = "L [SLIDEUP.R  :SPEED]
IF :KEY = "R [SLIDEDOWN.R  :SPEED]
END
```

In order to vary "speed" Jim simply needs to add the keys to increase and decrease speed:

```
IF :KEY = "F [MAKE "SPEED :SPEED + 1]
IF :KEY = "S [MAKE "SPEED :SPEED - 1]
```

The amount of turn, in this case the amount of the slide, can be made proportionate to the speed of the car. This was a principle we discussed with another group of boys last year. In thinking about driving a car, one realizes that turning the wheel has a greater effect at high speed than at lower speed. One can express this in Logo like this:

```
IF :KEY = "L [SLIDEUP.R :SPEED + (:SPEED * .07)]
```

The number to multiply, .07, can be manipulated to produce the desired effect.

DRIVE can also be changed to simulate friction:

```
TO DRIVE :SPEED
SLIDE.R  :SPEED
IF KEY? [DO READCHAR]
DRIVE :SPEED * .99
END
```

Jim's response to what we have done is the clearest instance yet of appropriation of previous ideas, in this case the slides, and the application of them in a way that we had certainly not expected, but which adds a very interesting new dimension to this work.

This project has certainly been a rewarding one for both teachers and students. Our agenda of programming reasonably accurate simulations of motion has fitted neatly into projects the boys were already highly motivated to develop. The project has also been a wonderful learning situation for us all. Each boy's new idea has caused us to draw upon our own resources, to rethink our own thinking and to take advantage of the many opportunities to talk about motion problems in a context which was important to the boys. In the end, we all have learned much more than we ever imagined.